

Multi-core Intel® Processor Architecture: Software Development Tools

Jason Plumb

**Parallel & Distributed Solutions Division
Intel Corporation**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

All dates provided are subject to change without notice.

* Other names and brands may be claimed as the property of others.

Copyright © 2005, Intel Corporation.

Agenda

- Parallel Processing
- Thread Level Parallelism
 - Programming
 - Analysis
 - Optimization

Intel® Architecture

Instruction Level Parallelism

Out-of-order instruction pipeline

Multiple execution units

Data Level Parallelism

MMX™

Streaming SIMD Extensions (SSE)

Streaming SIMD Extensions 2 (SSE2)

Streaming SIMD Extensions 3 (SSE3)

Thread Level Parallelism

Hyper-Threading Technology (HT Technology)

Multi-core Intel processor architecture

Intel® Architecture

Instruction Level Parallelism

Out-of-order instruction pipeline

Multiple execution units

Data Level Parallelism

MMX™

Streaming SIMD Extensions (SSE)

Streaming SIMD Extensions 2 (SSE2)

Streaming SIMD Extensions 3 (SSE3)

Thread Level Parallelism

Hyper-Threading Technology (HT Technology)

Multi-core Intel processor architecture

Intel® Architecture

Instruction Level Parallelism

Out-of-order instruction pipeline

Multiple execution units

Data Level Parallelism

MMX™

Streaming SIMD Extensions (SSE)

Streaming SIMD Extensions 2 (SSE2)

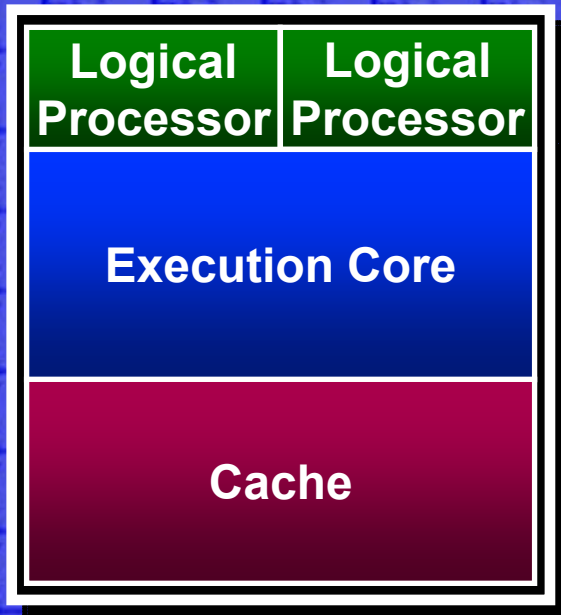
Streaming SIMD Extensions 3 (SSE3)

Thread Level Parallelism

Hyper-Threading Technology (HT Technology)

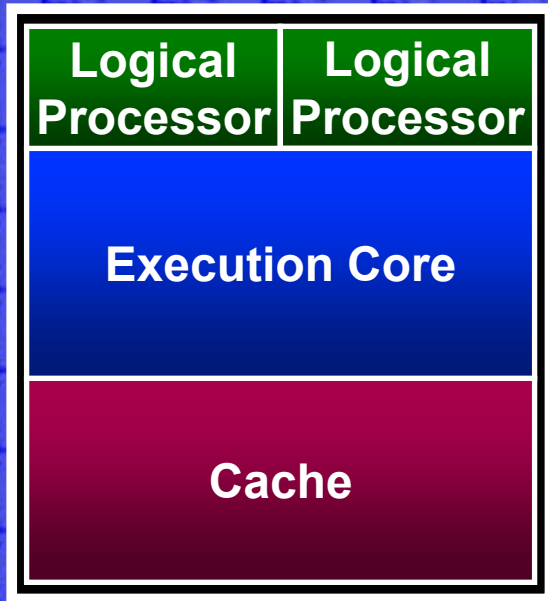
Multi-core Intel processor architecture

Hyper-Threading Technology (HT Technology) & Multi-core Intel processor architecture

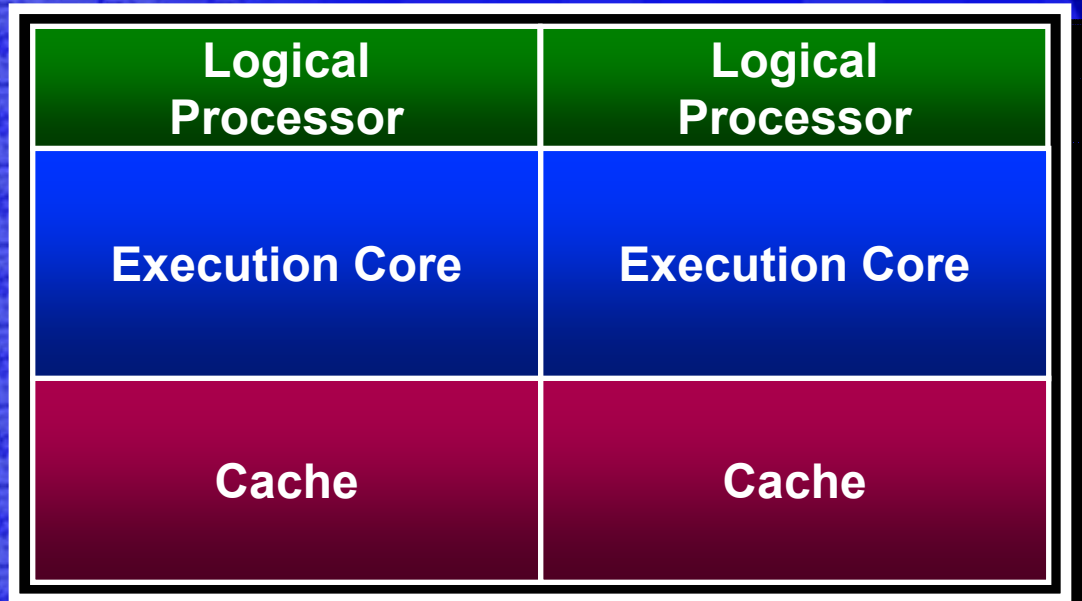


Intel® Xeon™/Pentium® 4
processor supporting
Hyper-Threading Technology

Hyper-Threading Technology (HT Technology) & Multi-core Intel processor architecture



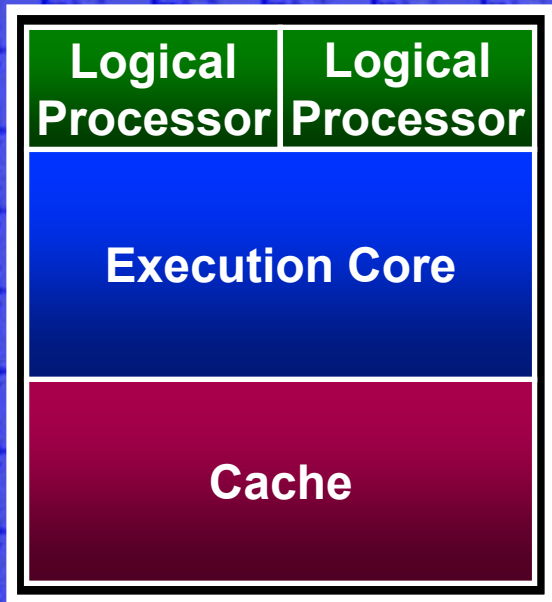
Intel® Xeon™/Pentium® 4
processor supporting
Hyper-Threading Technology



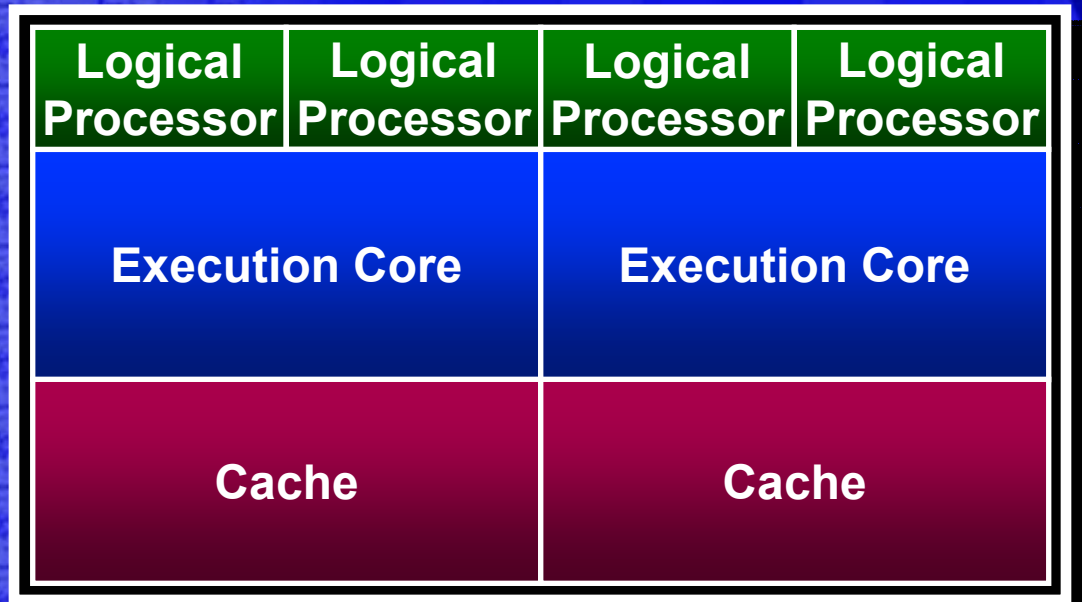
Pentium® D



Hyper-Threading Technology (HT Technology) & Multi-core Intel processor architecture



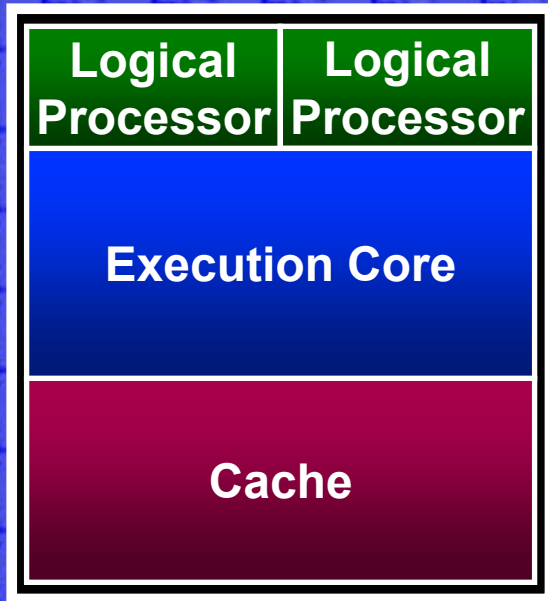
Intel® Xeon™/Pentium® 4
processor supporting
Hyper-Threading Technology



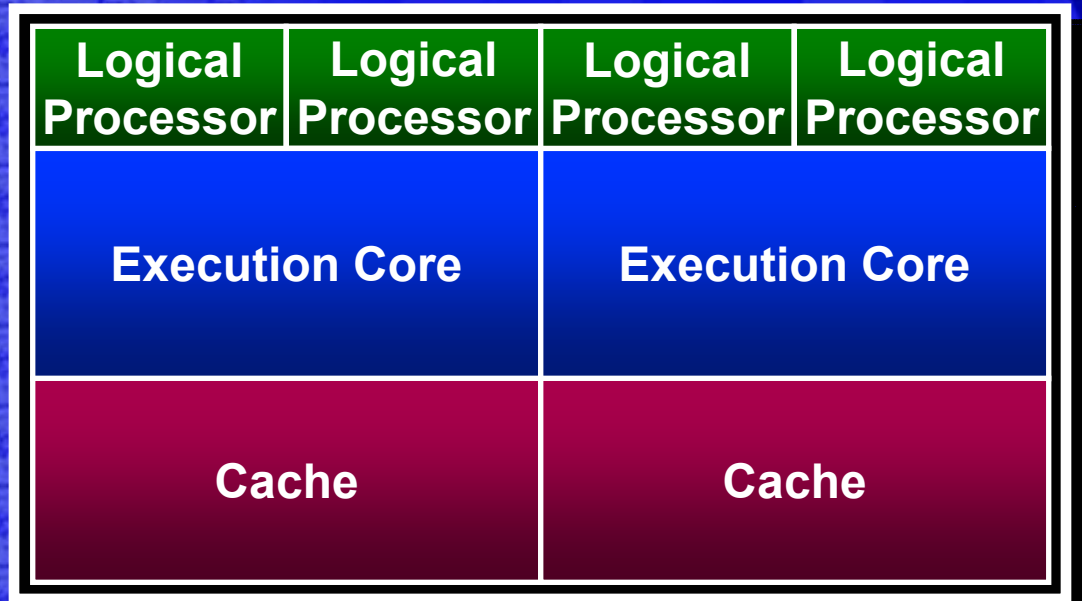
Pentium® processor Extreme Edition



Hyper-Threading Technology (HT Technology) & Multi-core Intel processor architecture



Intel® Xeon™/Pentium® 4 processor supporting Hyper-Threading Technology



Pentium® processor Extreme Edition

**Multiple execution cores
ramping across Intel platforms**

Programming Model



**Modern operating systems
load programs as processes**

**A process starts executing at
its entry point as a thread**

**Threads can create other
threads within the process**

**All threads within a process
share code & data segments**

**Threads have two scheduler
states (active, inactive)**

**OS schedules active threads to
execute on logical processors**

Programming Model



A diagram showing a large white rectangle representing a process. Inside the top-left corner of this rectangle is a smaller yellow rectangle. Inside the yellow rectangle, the word *thread* is written in black italicized font, and the text *main()* is written in red italicized font.

thread
main()

Modern operating systems
load programs as processes

A process starts executing at
its entry point as a thread

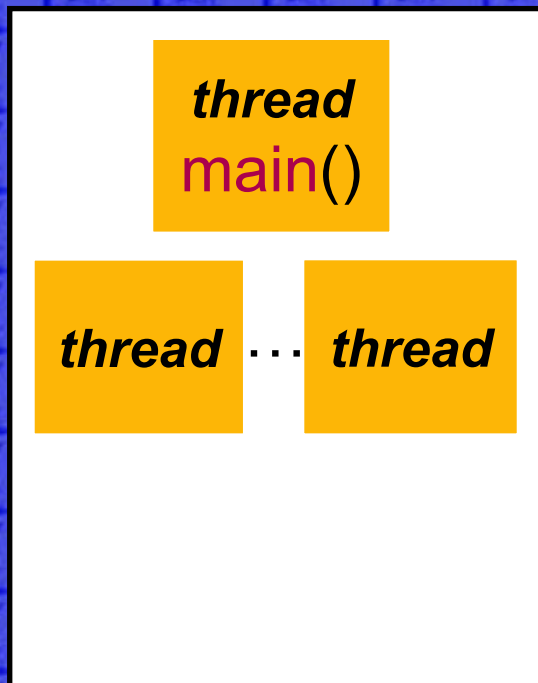
Threads can create other
threads within the process

All threads within a process
share code & data segments

Threads have two scheduler
states (active, inactive)

OS schedules active threads to
execute on logical processors

Programming Model



Modern operating systems load programs as processes

A process starts executing at its entry point as a thread

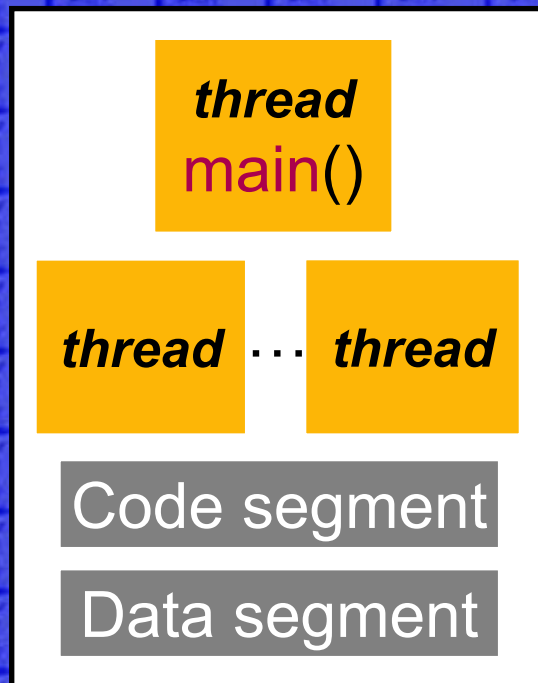
Threads can create other threads within the process

All threads within a process share code & data segments

Threads have two scheduler states (active, inactive)

OS schedules active threads to execute on logical processors

Programming Model



Modern operating systems load programs as processes

A process starts executing at its entry point as a thread

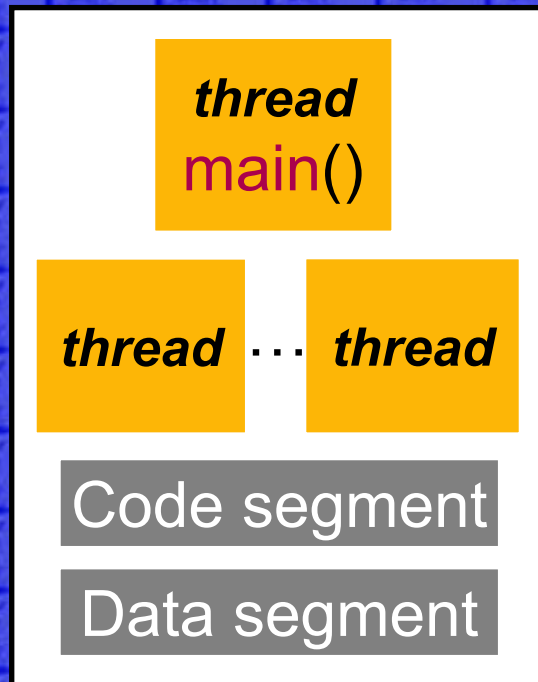
Threads can create other threads within the process

All threads within a process share code & data segments

Threads have two scheduler states (active, inactive)

OS schedules active threads to execute on logical processors

Programming Model



Modern operating systems load programs as processes

A process starts executing at its entry point as a thread

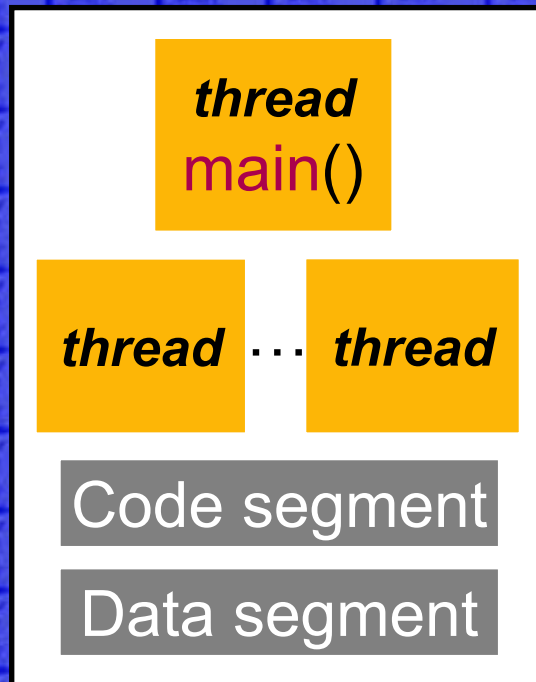
Threads can create other threads within the process

All threads within a process share code & data segments

Threads have two scheduler states (active, inactive)

OS schedules active threads to execute on logical processors

Programming Model



Modern operating systems load programs as processes

A process starts executing at its entry point as a thread

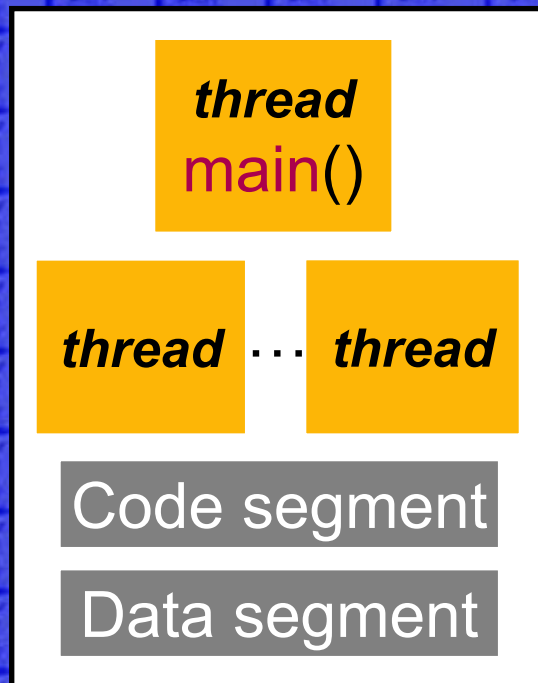
Threads can create other threads within the process

All threads within a process share code & data segments

Threads have two scheduler states (active, inactive)

OS schedules active threads to execute on logical processors

Programming Model



Modern operating systems load programs as processes

A process starts executing at its entry point as a thread

Threads can create other threads within the process

All threads within a process share code & data segments

Threads have two scheduler states (active, inactive)

OS schedules active threads to execute on logical processors

Decompose algorithms across parallel compute threads

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    for (long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```


Example: Prime Number Generation

2
3 3
5 3,5
7 3,5,7
9 3
11 3,5,7,9,11
13 3,5,7,9,11,13
15 3
17 3,5,7,9,11,13,15,17

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    for (long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

2
3 3
5 3,5
7 3,5,7
9 3
11 3,5,7,9,11
13 3,5,7,9,11,13
15 3
17 3,5,7,9,11,13,15,17

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }

    printf( "Found %d primes\n", number_of_primes );
}
```


Example: Prime Number Generation

2
3 3
5 3,5
7 3,5,7
9 3
11 3,5,7,9,11
13 3,5,7,9,11,13
15 3
17 3,5,7,9,11,13,15,17

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }

    printf( "Found %d primes\n", number_of_primes );
}
```

```
Determining primes from 1-18
Found 7 primes
Press any key to continue_
```

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 1000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case

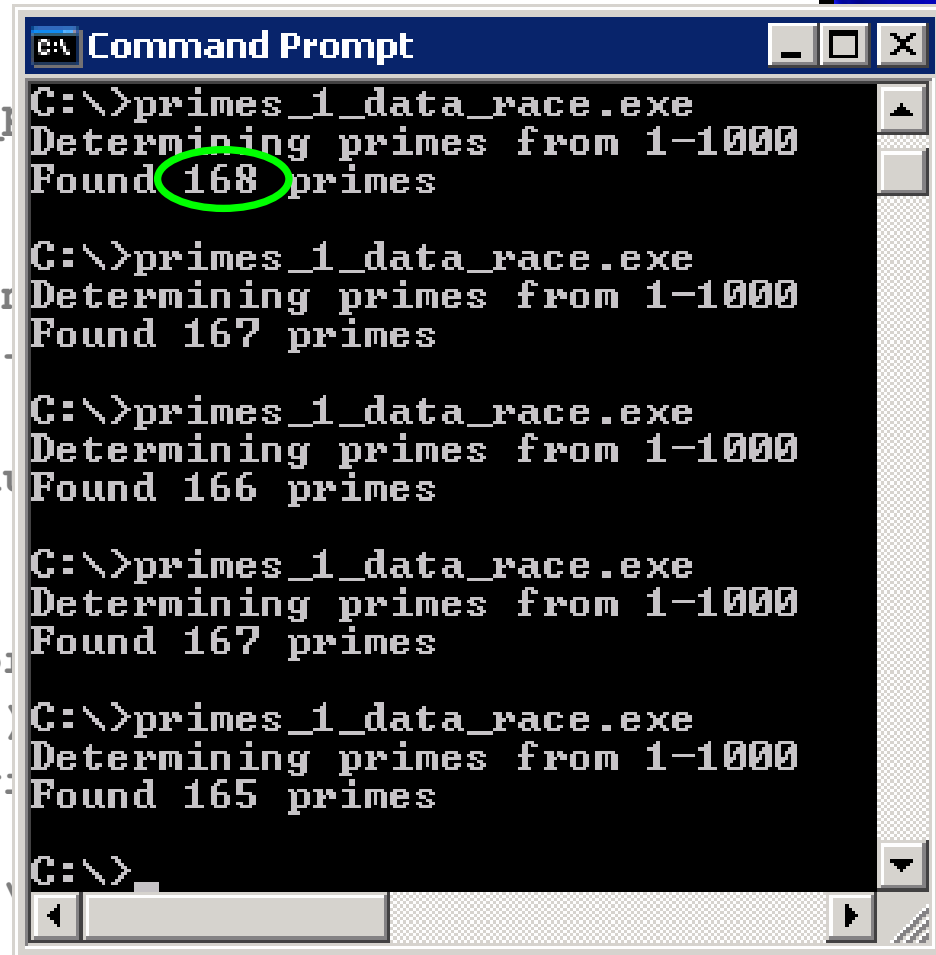
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```


Example: Prime Number Generation

```
#include <stdio.h>
const long N = 1000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
#pragma omp parallel for
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 1000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-1000\n" );
    primes[ number_of_primes ] = 2;
    number_of_primes++;
    #pragma omp parallel for
    for ( long number = 3; number <= 1000; number++ )
    {
        long factor = 3;
        while ( number % factor != 0 )
            factor++;
        if ( factor == number )
            primes[ number_of_primes ] = number;
            number_of_primes++;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```



```
Command Prompt
C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 167 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 166 primes

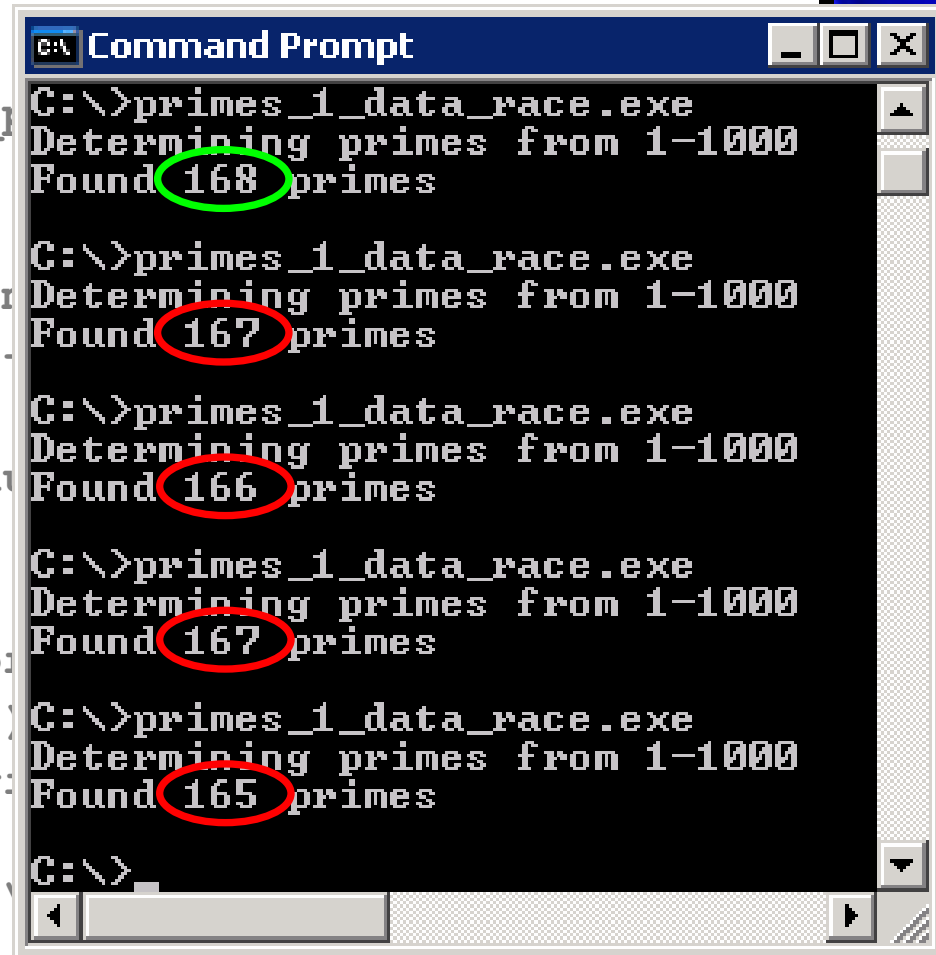
C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 167 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 165 primes

C:\>
```


Example: Prime Number Generation

```
#include <stdio.h>
const long N = 1000;
long primes[N], number_of_primes;
main()
{
    printf( "Determining primes from 1-1000\n" );
    primes[ number_of_primes ] = 0;
    #pragma omp parallel for
    for ( long number = 3; number <= 1000; number += 2 )
    {
        long factor = 3;
        while ( number % factor != 0 )
            factor += 1;
        if ( factor == number )
            primes[ number_of_primes ] = number;
        number_of_primes++;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```



```
Command Prompt
C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 167 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 166 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 167 primes

C:\>primes_1_data_race.exe
Determining primes from 1-1000
Found 165 primes

C:\>
```

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 1000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
#pragma omp parallel for
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

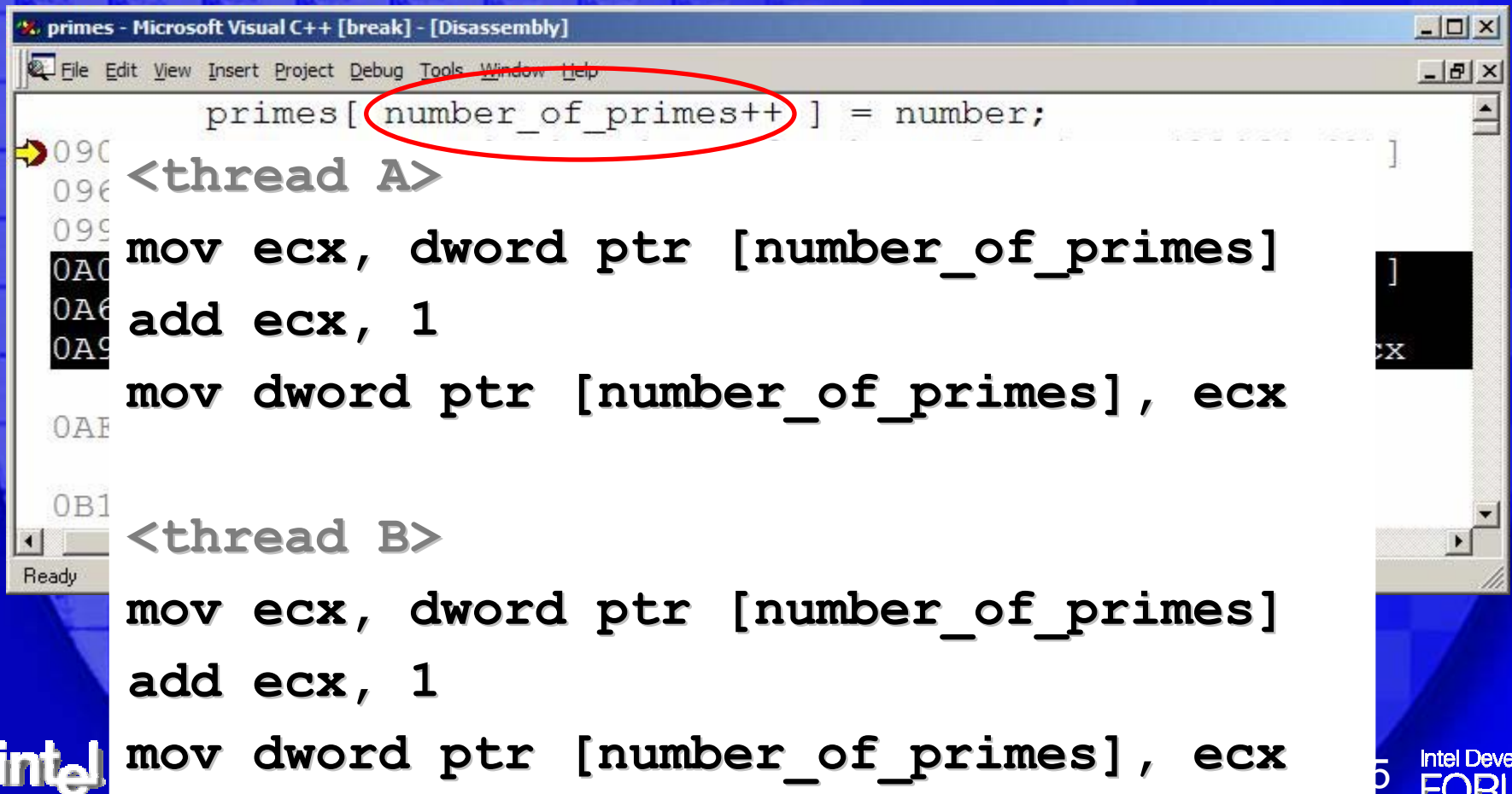

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 100000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    #pragma omp parallel for
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

Pre-emptive context switching and shared memory conflicts (Data Races)

```
primes - Microsoft Visual C++ [break] - [Disassembly]
File Edit View Insert Project Debug Tools Window Help
primes[ number_of_primes++ ] = number;
090  mov     edx, dword ptr [number_of_primes (00464ad0)]
096  mov     eax, dword ptr [number]
099  mov     dword ptr [edx*4+403050h], eax
0A0  mov     ecx, dword ptr [number_of_primes (00464ad0)]
0A6  add     ecx, 1
0A9  mov     dword ptr [number_of_primes (00464ad0)], ecx
    }
0AF  jmp     main+59h (00401059)
printf( "Found %d primes\n", number_of_primes );
0B1  mov     esi, esp
Ready
```


Pre-emptive context switching and shared memory conflicts (Data Races)

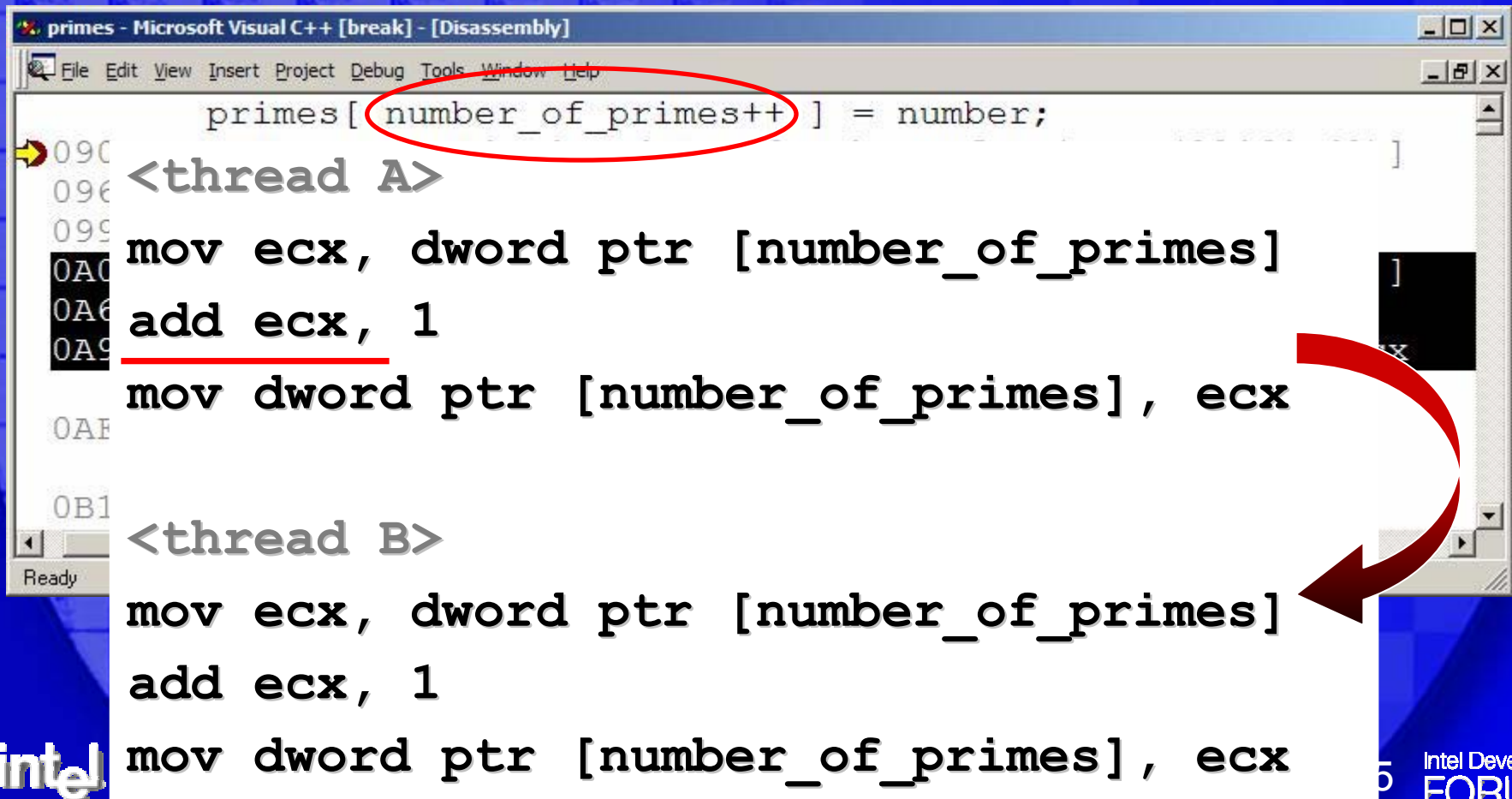


The screenshot shows a disassembler window for a program named 'primes'. The source code at the top is `primes[number_of_primes++] = number;`, with `number_of_primes++` circled in red. Below, the assembly for two threads is shown. Thread A increments `number_of_primes` from 0 to 1. Thread B then increments it from 1 to 2. This illustrates a data race where the final value of `number_of_primes` is 2, even though two different prime numbers were intended to be stored.

```
primes - Microsoft Visual C++ [break] - [Disassembly]
File Edit View Insert Project Debug Tools Window Help
primes[ number_of_primes++ ] = number;

0900 <thread A>
0906 mov ecx, dword ptr [number_of_primes]
0909 add ecx, 1
0A00 mov dword ptr [number_of_primes], ecx
0A06
0A09
0AE0
0B10 <thread B>
0B16 mov ecx, dword ptr [number_of_primes]
0B19 add ecx, 1
0B1C mov dword ptr [number_of_primes], ecx
```

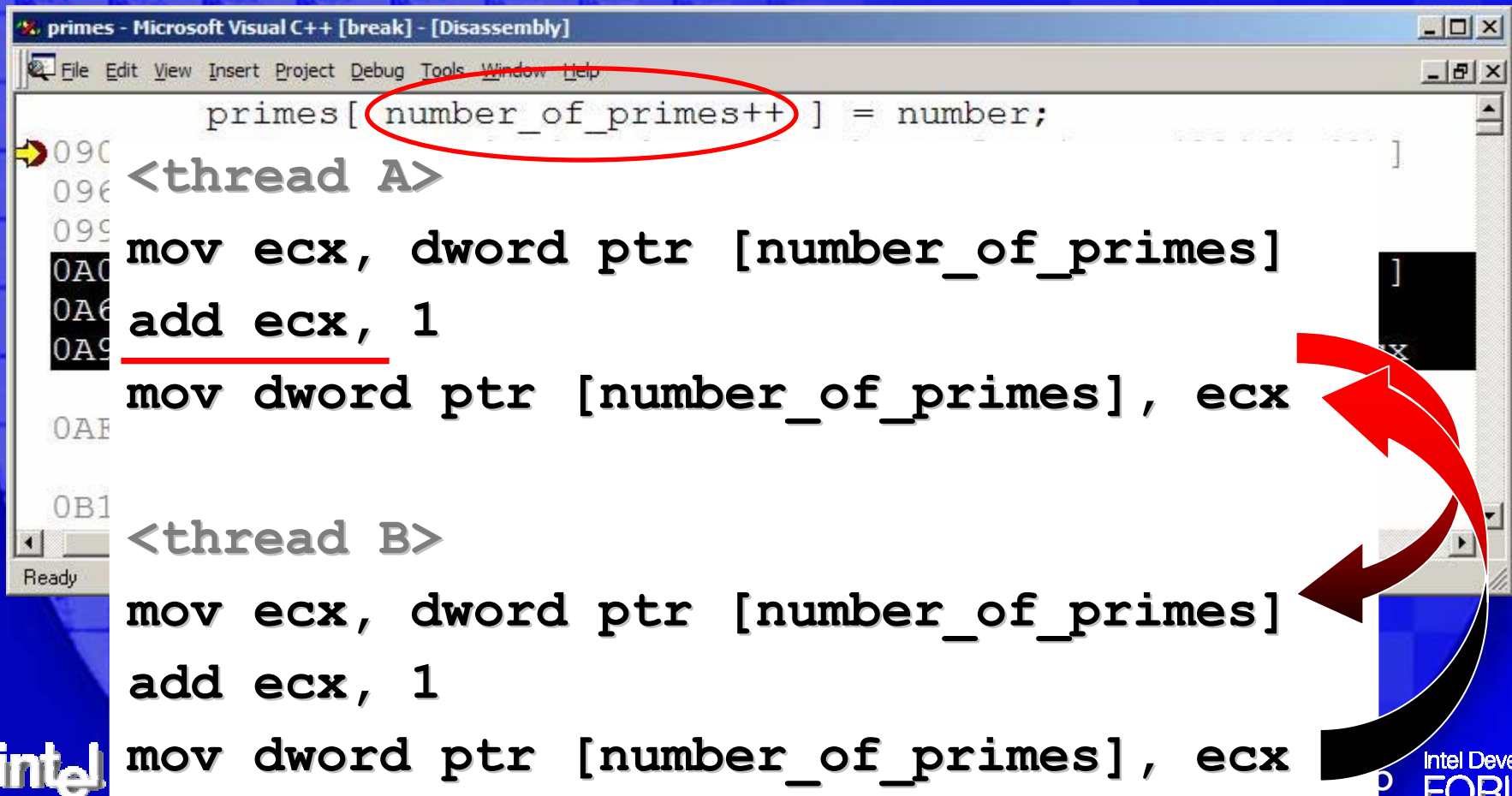
Pre-emptive context switching and shared memory conflicts (Data Races)



```
primes - Microsoft Visual C++ [break] - [Disassembly]
File Edit View Insert Project Debug Tools Window Help
primes[ number_of_primes++ ] = number;

09C0 <thread A>
09C6
09C9 mov ecx, dword ptr [number_of_primes]
0A00
0A06 add ecx, 1
0A09
0AE0 mov dword ptr [number_of_primes], ecx
0AEF
0B10 <thread B>
0B16
0B19 mov ecx, dword ptr [number_of_primes]
0B20
0B23 add ecx, 1
0B26
0B29 mov dword ptr [number_of_primes], ecx
```

Pre-emptive context switching and shared memory conflicts (Data Races)



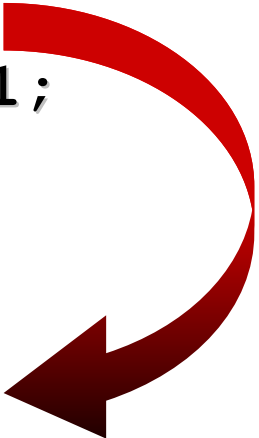
```
primes - Microsoft Visual C++ [break] - [Disassembly]
File Edit View Insert Project Debug Tools Window Help
primes[ number_of_primes++ ] = number;
<thread A>
09C0 mov ecx, dword ptr [number_of_primes]
09C6 add ecx, 1
09C9 mov dword ptr [number_of_primes], ecx
09AE
0B10 <thread B>
0B12 mov ecx, dword ptr [number_of_primes]
0B14 add ecx, 1
0B16 mov dword ptr [number_of_primes], ecx
```


Data Race

```
int s;  
threadA() {  
    ...  
    s = 1;  
    int A = s+1;  
    ...  
threadB() {  
    ...  
    s = 2;  
    int B = s+1;  
    ...
```

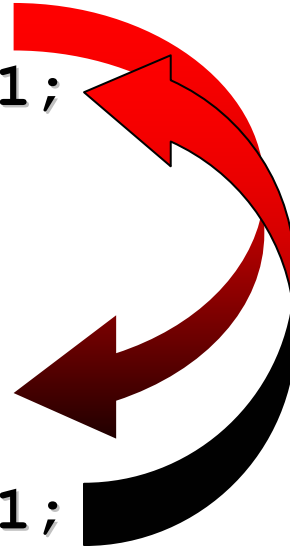
Data Race

```
int s;  
threadA() {  
    ...  
    s = 1;  
    int A = s+1;  
    ...  
threadB() {  
    ...  
    s = 2;  
    int B = s+1;  
    ...
```



Data Race

```
int s;  
threadA() {  
    ...  
    s = 1;  
    int A = s+1;  
    ...  
threadB() {  
    ...  
    s = 2;  
    int B = s+1;  
    ...
```

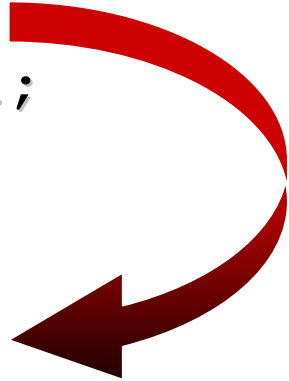


Synchronized Resource Sharing

```
int s; mutex m;  
threadA() {  
    lock(&m);  
    s = 1;  
    int A = s+1;  
    unlock(&m);  
threadB() {  
    lock(&m);  
    s = 2;  
    int B = s+1;  
    unlock(&m);
```

Synchronized Resource Sharing

```
int s; mutex m;  
threadA() {  
    lock(&m);  
    s = 1;  
    int A = s+1;  
    unlock(&m);  
threadB() {  
    lock(&m);  
    s = 2;  
    int B = s+1;  
    unlock(&m);
```



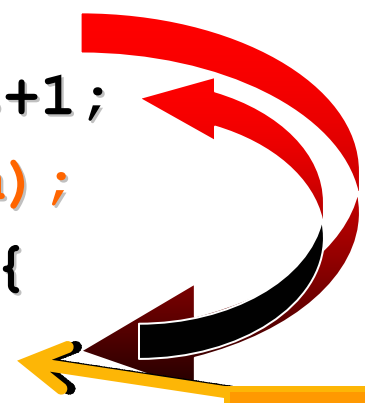
Synchronized Resource Sharing

```
int s; mutex m;  
threadA() {  
    lock(&m);  
    s = 1;  
    int A = s+1;  
    unlock(&m);  
threadB() {  
    lock(&m);  
    s = 2;  
    int B = s+1;  
    unlock(&m);
```

Does not
return until
acquired

Synchronized Resource Sharing

```
int s; mutex m;  
threadA() {  
    lock(&m);  
    s = 1;  
    int A = s+1;  
    unlock(&m);  
threadB() {  
    lock(&m);  
    s = 2;  
    int B = s+1;  
    unlock(&m);
```



**Does not
return until
acquired**

Intel® Threading Tools

Intel® Thread Checker



Pinpoint notorious threading bugs like data races, stalls and deadlocks

Thread Level Parallelism

Intel® Thread Checker

VTune™ Performance Analyzer

Intel® Thread Checker

Intel® Thread Checker

VTune™ Performance Analyzer

Intel® Thread Checker

Primes.exe*

***Compilation options: /Zi /O0**

***Link options: /debug /fixed:no**

Intel® Thread Checker

Primes.exe*

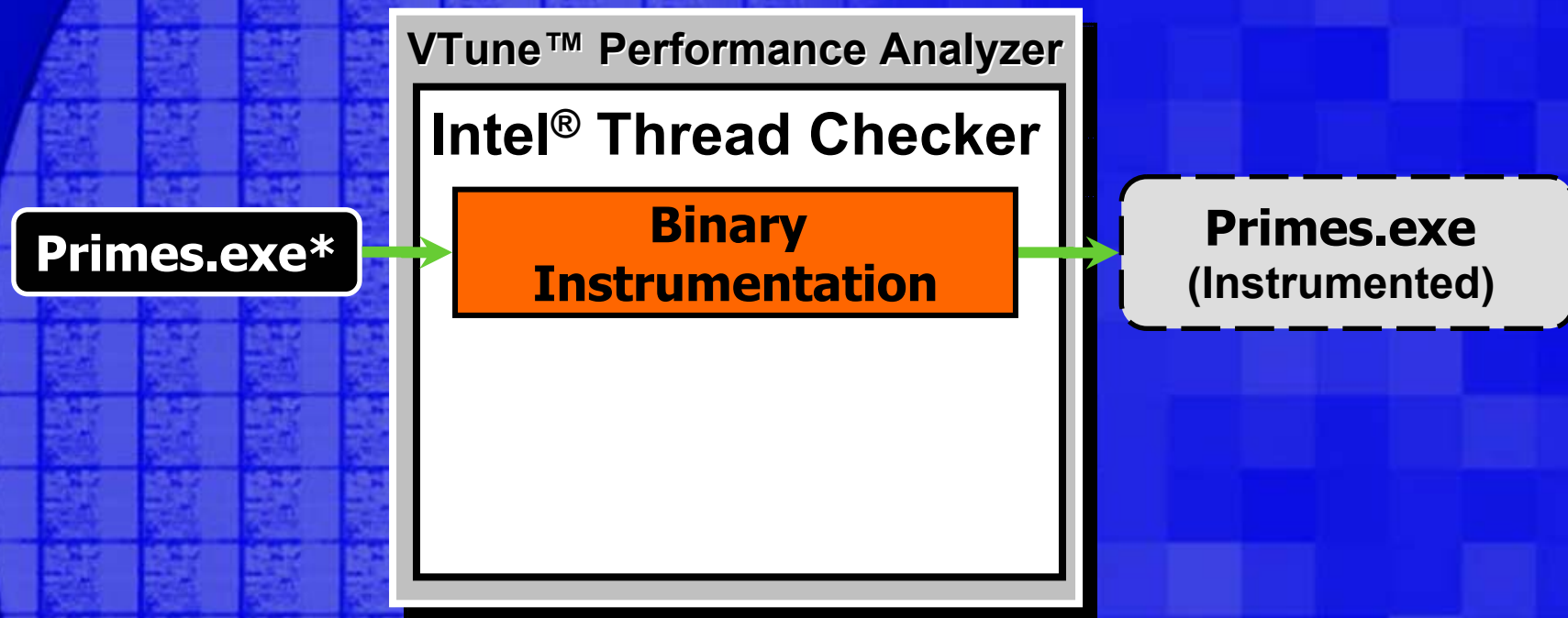
VTune™ Performance Analyzer

Intel® Thread Checker

***Compilation options: /Zi /O0**

***Link options: /debug /fixed:no**

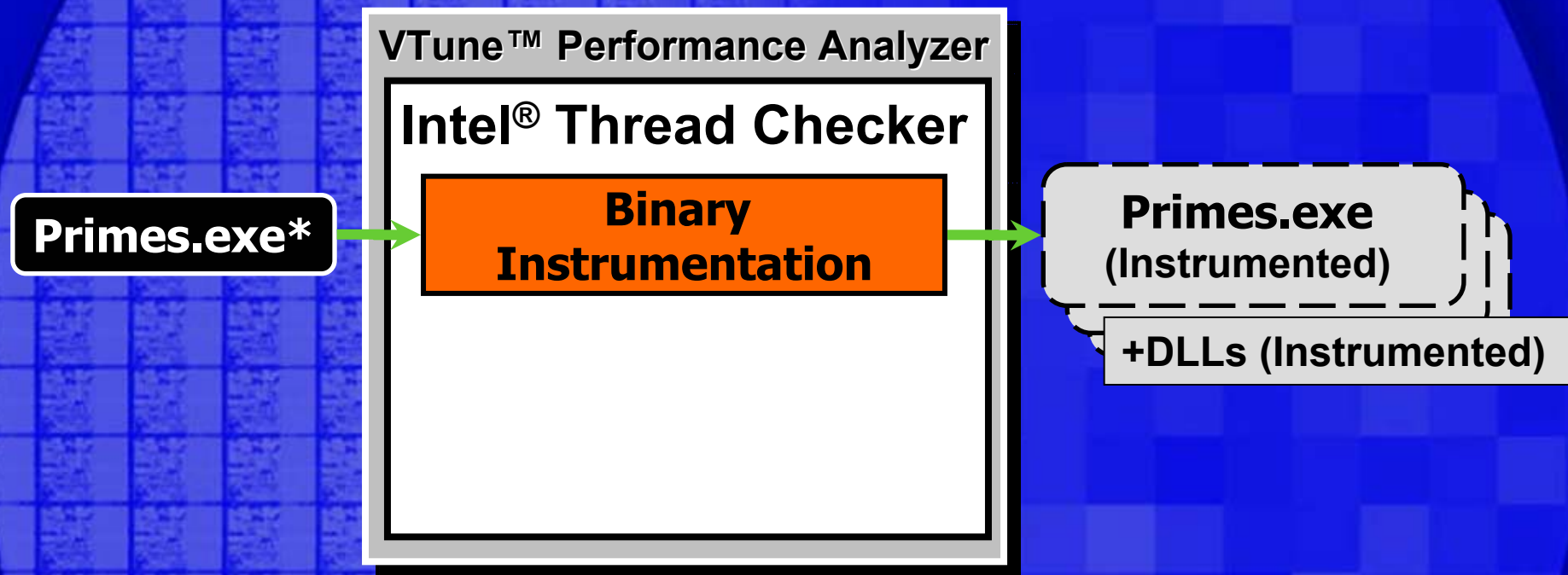
Intel® Thread Checker



***Compilation options: /Zi /O0**

***Link options: /debug /fixed:no**

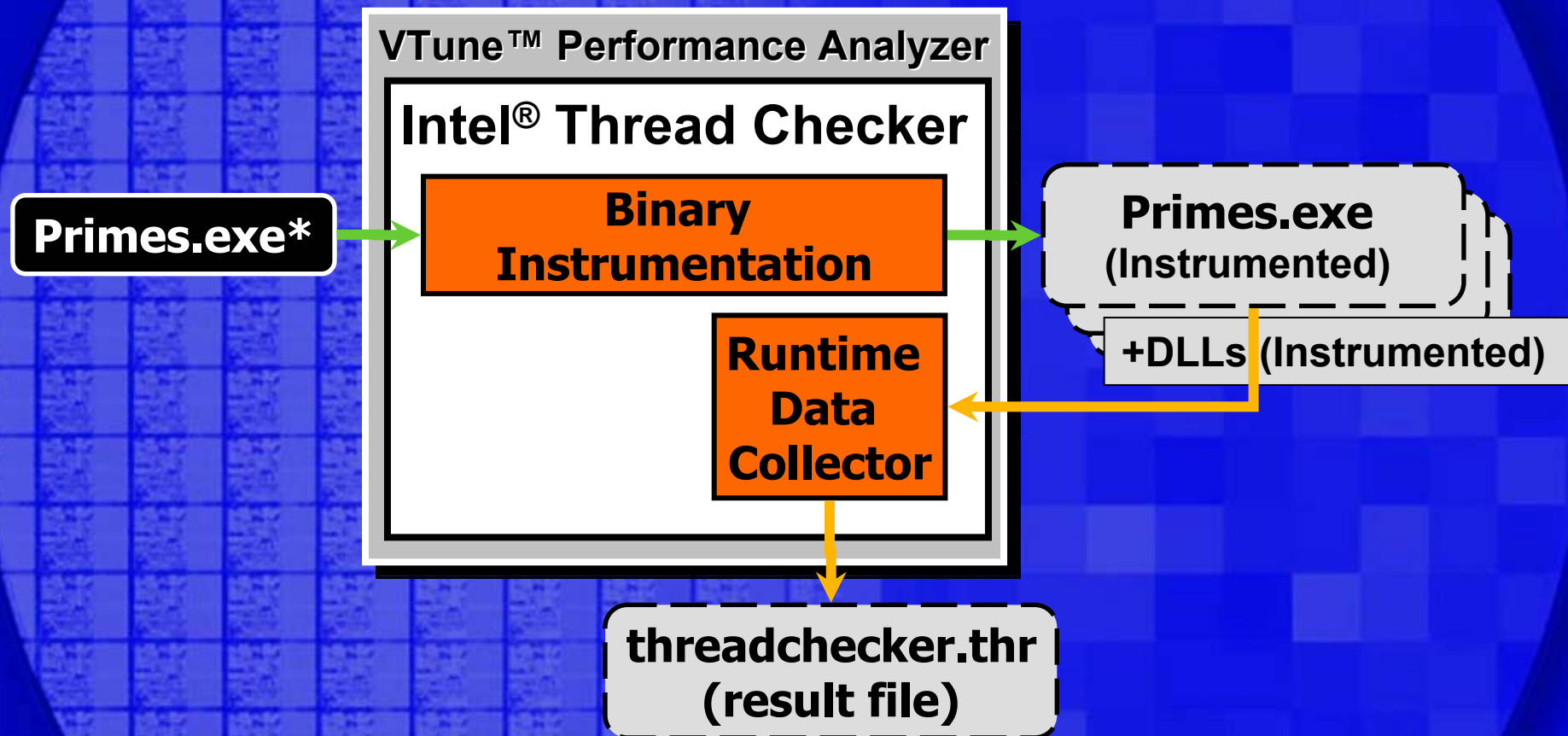
Intel® Thread Checker



***Compilation options: /Zi /O0**

***Link options: /debug /fixed:no**

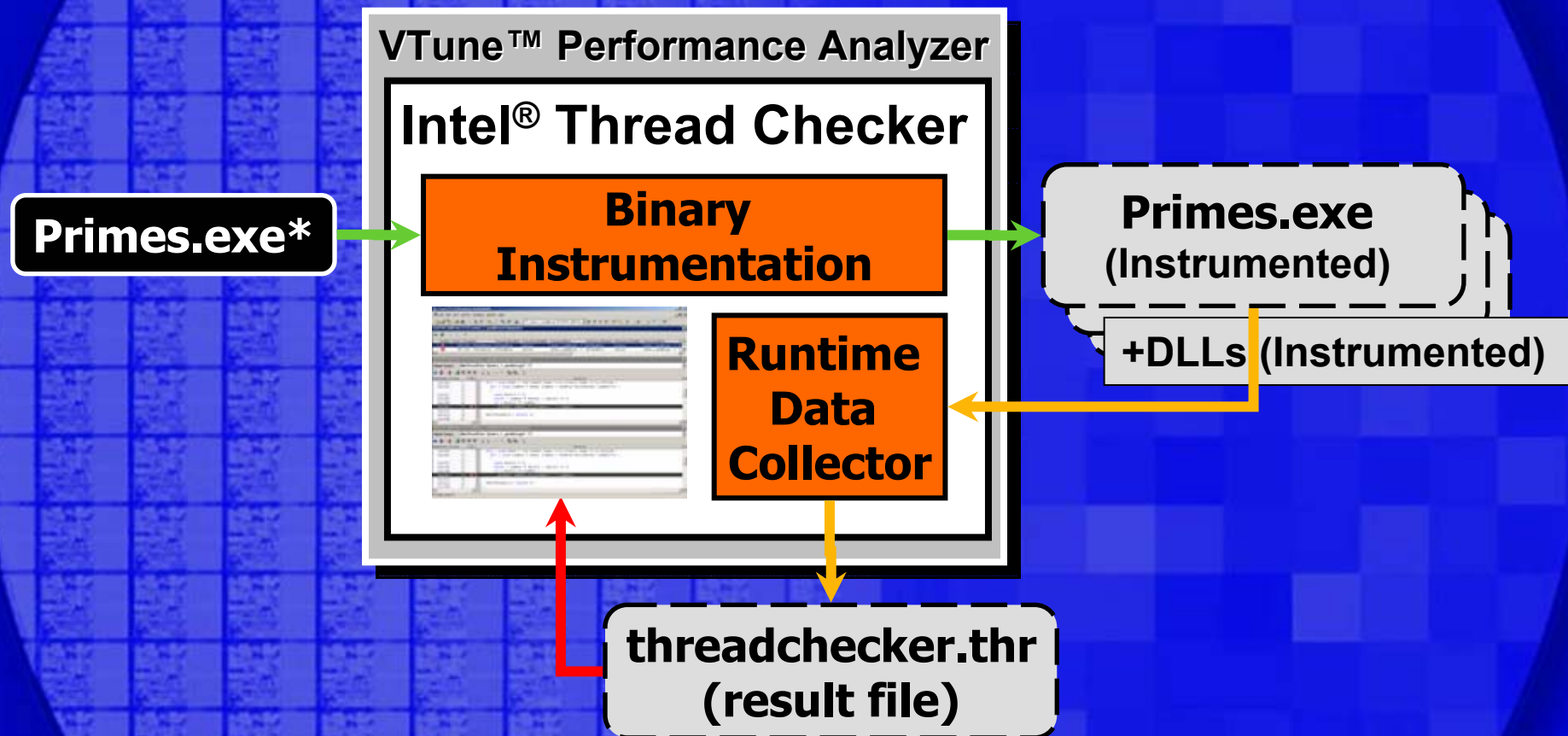
Intel® Thread Checker



***Compilation options: /Zi /O0**

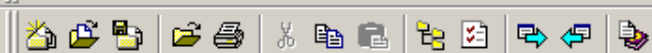
***Link options: /debug /fixed:no**

Intel® Thread Checker



***Compilation options: /Zi /O0**

***Link options: /debug /fixed:no**



TC: 2_parallel_openmp.exe (08:33 PM, 2005 Mar 22)

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe): Diagnostics



Severity	Counts	Description	Context [Best]
1	1	Write -> Read data-race	"2_parallel_openmp.cpp": 17
2	1	Write -> Write data-race	"2_parallel_openmp.cpp": 17

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe) (ID=0)2nd Access

Stack Trace: main "2_parallel_openmp.cpp": 14



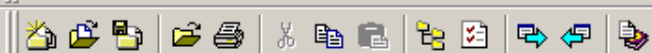
Address	Line	Source
0x1085	8	#pragma omp parallel for
0x116C	9	for (long number = 3; number <= N; number += 2)
	10	{
0x11DC	11	long factor = 3;
0x11E3	12	while (number % factor) factor += 2;
0x1212	13	if (factor == number)
0x121C	14	primes[number_of_primes++] = number;
	15	}
0x1237	16	printf("Found %d primes\n", number_of_primes);

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe) (ID=0)1st Access

Stack Trace: main "2_parallel_openmp.cpp": 14



Address	Line	Source
0x1085	8	#pragma omp parallel for
0x116C	9	for (long number = 3; number <= N; number += 2)
	10	{
0x11DC	11	long factor = 3;
0x11E3	12	while (number % factor) factor += 2;
0x1212	13	if (factor == number)
0x121C	14	primes[number_of_primes++] = number;
	15	}
0x1237	16	printf("Found %d primes\n", number_of_primes);



TC: 2_parallel_openmp.exe (08:33 PM, 2005 Mar 22)

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe): Diagnostics



Severity	Counts	Description	Context (Best)
1		Write -> Read data-race	"2_parallel_openmp.cpp": 17
2	1	Write -> Write data-race	"2_parallel_openmp.cpp": 17

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe) (ID=0)2nd Access

Stack Trace: main "2_parallel_openmp.cpp": 14



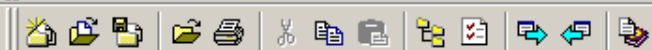
Address	Line		Source
0x1085	8		#pragma omp parallel for
0x116C	9		for (long number = 3; number <= N; number += 2)
	10		{
0x11DC	11		long factor = 3;
0x11E3	12		while (number % factor) factor += 2;
0x1212	13		if (factor == number)
0x121C	14		primes[number_of_primes++] = number;
	15		}
0x1237	16		printf("Found %d primes\n", number_of_primes);

08:35 PM, 2005 Mar 22 (TC: 2_parallel_openmp.exe) (ID=0)1st Access

Stack Trace: main "2_parallel_openmp.cpp": 14



Address	Line		Source
0x1085	8		#pragma omp parallel for
0x116C	9		for (long number = 3; number <= N; number += 2)
	10		{
0x11DC	11		long factor = 3;
0x11E3	12		while (number % factor) factor += 2;
0x1212	13		if (factor == number)
0x121C	14		primes[number_of_primes++] = number;
	15		}
0x1237	16		printf("Found %d primes\n", number_of_primes);



TC: 2_parallel_omp.exe (08:33 PM, 2005 Mar 22)

08:35 PM, 2005 Mar 22 (TC: 2_parallel_omp.exe): Diagnostics



Severity	Counts	Description	Context [Best]
1		Write - Read data-race	"2_parallel_omp.cpp": 17
2	1	Write -> Write data-race	"2_parallel_omp.cpp": 17

08:35 PM, 2005 Mar 22 (TC: 2_parallel_omp.exe) (ID=0) 2nd Access

Stack Trace: main "2_parallel_omp.cpp": 14



Address	Line	Source
0x1085	8	#pragma omp parallel for
0x116C	9	for (long number = 3; number <= N; number += 2)
	10	{
0x11DC	11	long factor = 3;
0x11E3	12	while (number % factor) factor += 2;
0x1212	13	if (factor == number)
0x121C	14	primes number_of_primes++ = number;
	15	}
0x1237	16	printf("Found %d primes\n", number_of_primes);

08:35 PM, 2005 Mar 22 (TC: 2_parallel_omp.exe) (ID=0) 1st Access

Stack Trace: main "2_parallel_omp.cpp": 14



Address	Line	Source
0x1085	8	#pragma omp parallel for
0x116C	9	for (long number = 3; number <= N; number += 2)
	10	{
0x11DC	11	long factor = 3;
0x11E3	12	while (number % factor) factor += 2;
0x1212	13	if (factor == number)
0x121C	14	primes number_of_primes++ = number;
	15	}
0x1237	16	printf("Found %d primes\n", number_of_primes);

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 100000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    #pragma omp parallel for
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )

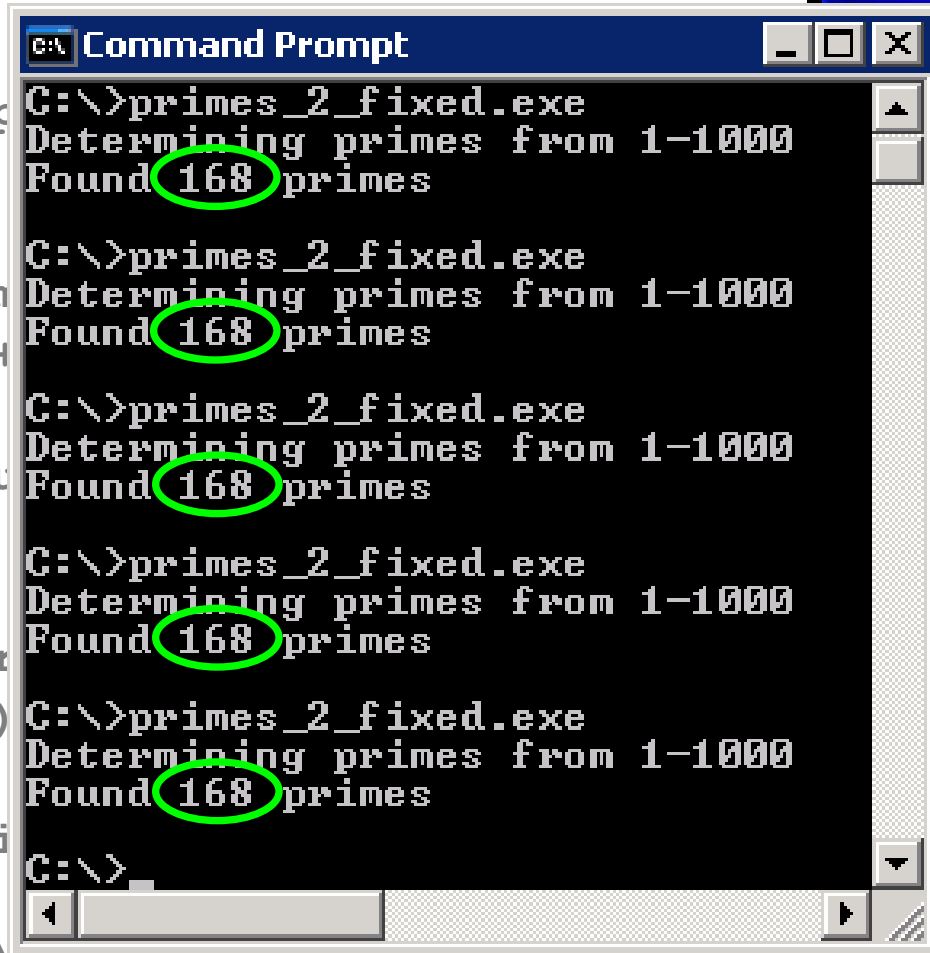
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 100000;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    #pragma omp parallel for
    for ( long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            #pragma omp critical
                primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

```
#include <stdio.h>
const long N = 100000;
long primes[N], number_of_p
main()
{
    printf( "Determining prim
    primes[ number_of_primes+
#pragma omp parallel for
    for ( long number = 3; nu
    {
        long factor = 3;
        while ( number % factor
        if ( factor == number )
#pragma omp critical
        primes[ number_of_pri
    }
    printf( "Found %d primes\n", number_of_primes );
}
```



```
C:\>primes_2_fixed.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_2_fixed.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_2_fixed.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_2_fixed.exe
Determining primes from 1-1000
Found 168 primes

C:\>primes_2_fixed.exe
Determining primes from 1-1000
Found 168 primes

C:\>
```


Intel® Threading Tools



Intel® Thread Checker

Pinpoint notorious threading bugs like data races, stalls and deadlocks

Intel® Thread Profiler

Visualize your threads over time

Identify synchronization objects that cause delays

OpenMP*, Win32* threads, POSIX* threads

Example: Prime Number Generation

```
#include <windows.h>
#include <stdio.h>

const long N = 1000;
long primes[N], number_of_primes = 0;

typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
{
    THREAD_ARGS* p = (THREAD_ARGS*)pThreadArgs;
    for ( long base = 3+p->seed; base <= p->limit; base += p->stride )
        for ( long number = base; number < base+p->blocksize; number+=2 )
        {
            long factor = 3;
            while ( number % factor ) factor += 2;
            if ( factor == number )
                primes[ number_of_primes++ ] = number;
        }
    ExitThread(0); return 0;
}

main()
{
    // Decompose computation across processors
    SYSTEM_INFO info; GetSystemInfo(&info);
    int num_threads = info.dwNumberOfProcessors;
    HANDLE* pThreadHandles = (HANDLE*)malloc( num_threads * sizeof(HANDLE) );
    THREAD_ARGS* pThreadArgs =
        (THREAD_ARGS*)malloc( num_threads * sizeof(THREAD_ARGS) );

    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case

    for ( int thread = 0; thread < num_threads; thread++ )
    {
        pThreadArgs[thread].blocksize = N / num_threads;
        pThreadArgs[thread].seed = thread * pThreadArgs[thread].blocksize;
        pThreadArgs[thread].limit = N;
        pThreadArgs[thread].stride =
            num_threads * pThreadArgs[thread].blocksize;

        pThreadHandles[thread] =
            CreateThread(0,0,MyThreadFunc, (LPVOID) (pThreadArgs+thread), 0,0);
    }
    WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );
    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

```
#include <windows.h>
#include <stdio.h>

const long N = 1000;
long primes[N], number_of_primes = 0;

typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
```

```
main()
```

```
{
```

```
// Decompose computation across processors
```

```
SYSTEM_INFO info; GetSystemInfo(&info);
```

```
int num_threads = info.dwNumberOfProcessors;
```

```
HANDLE* pThreadHandles =
```

```
(HANDLE*)malloc( num_threads * sizeof(HANDLE) );
```

```
THREAD_ARGS* pThreadArgs =
```

```
(THREAD_ARGS*)malloc( num_threads * sizeof(THREAD_ARGS) );
```

```
...
```

```
    pThreadHandles[thread] =
        CreateThread(0,0,MyThreadFunc, (LPVOID) (pThreadArgs+thread), 0,0);
}
WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );
printf( "Found %d primes\n", number_of_primes );
}
```


Example: Prime Number Generation

```
#include <windows.h>
#include <stdio.h>

const long N = 1000;
long primes[N], number_of_primes = 0;

typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
```

```
main()
{
    ...
    for ( int thread = 0; thread < num_threads; thread++ )
    {
        pThreadArgs[thread].blocksize = N / num_threads;
        pThreadArgs[thread].seed = thread * pThreadArgs[thread].blocksize;
        pThreadArgs[thread].limit = N;
        pThreadArgs[thread].stride = num_threads * pThreadArgs[thread].blocksize;

        pThreadHandles[thread] =
            CreateThread(0,0,MyThreadFunc,(LPVOID)(pThreadArgs+thread),0,0);
    }
    WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );

    printf( "Found %d primes\n", number_of_primes );
}
```

Example: Prime Number Generation

```

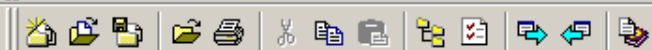
DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
{
    THREAD_ARGS* p = (THREAD_ARGS*)pThreadArgs;
    for ( long base = 3+p->seed; base <= p->limit; base += p->stride )
        for ( long number = base; number < base+p->blocksize; number+=2 )
        {
            long factor = 3;
            while ( number % factor ) factor += 2;
            if ( factor == number )
                primes[ number_of_primes++ ] = number;
        }
    pThreadArgs[thread].processed = thread;
    pThreadArgs[thread].limit = N;
    pThreadArgs[thread].stride = num_threads * pThreadArgs[thread].blocksize;

    pThreadHandles[thread] =
        CreateThread(0,0,MyThreadFunc,(LPVOID)(pThreadArgs+thread),0,0);
}

WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );

printf( "Found %d primes\n", number_of_primes );
}

```



TC: 2_parallel_win32.exe (08:44 PM, 2005 Mar)

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe): Diagnostics



ID	Severity	Counts	Description	Context [Best]
1		45	Write -> Read data-race	"2_parallel_win32.cpp": 8
2		45	Write -> Write data-race	"2_parallel_win32.cpp": 8

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1)2nd Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16



Address	Line		Source
0x105D	11		for (long number = base; number < base+p->blocksize; number+=2)
	12		{
0x108E	13		long factor = 3;
0x1095	14		while (number % factor) factor += 2;
0x10C4	15		if (factor == number)
0x10CE	16		primes[number of primes++] = number;
	17		}
0x10E9	18		ExitThread(0); return 0;
	19		}
	20		
	21		main()

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1)1st Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16



Address	Line		Source
0x108E	13		long factor = 3;
0x1095	14		while (number % factor) factor += 2;
0x10C4	15		if (factor == number)
0x10CE	16		primes[number of primes++] = number;
	17		}
0x10E9	18		ExitThread(0); return 0;
	19		}
	20		

File Edit View Activity Configure Window Help

TC: 2_parallel_win32.exe (08:44 PM, 2005 Mar)

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe): Diagnostics

ID	Severity	Counts	Description	Context [Best]
1	●	45	Write -> Read data-race	"2_parallel_win32.cpp": 8
2	●	45	Write -> Write data-race	"2_parallel_win32.cpp": 8

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1) 2nd Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16

Address	Line	Source
0x105D	11	for (long number = base; number < base+p->blocksize; number+=2)
	12	{
0x108E	13	long factor = 3;
0x1095	14	while (number % factor) factor += 2;
0x10C4	15	if (factor == number)
0x10CE	16	primes[number of primes++] = number;
	17	}
0x10E9	18	ExitThread(0); return 0;
	19	}
	20	
	21	main()

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1) 1st Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16

Address	Line	Source
0x108E	13	long factor = 3;
0x1095	14	while (number % factor) factor += 2;
0x10C4	15	if (factor == number)
0x10CE	16	primes[number of primes++] = number;
	17	}
0x10E9	18	ExitThread(0); return 0;
	19	}
	20	

Context Definition 1st Access Stack Traces

File Edit View Activity Configure Window Help

TC: 2_parallel_win32.exe (08:44 PM, 2005 Mar)

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe): Diagnostics

ID	Severity	Counts	Description	Context [Best]
1	●	45	Write - Read data-race	"2_parallel_win32.cpp": 8
2	●	45	Write -> Write data-race	"2_parallel_win32.cpp": 8

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1) 2nd Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16

Address	Line	Source
0x105D	11	for (long number = base; number < base+p->blocksize; number+=2)
	12	{
0x108E	13	long factor = 3;
0x1095	14	while (number % factor) factor += 2;
0x10C4	15	if (factor == number)
0x10CE	16	primes[number of primes++] = number;
	17	}
0x10E9	18	ExitThread(0); return 0;
	19	}
	20	
	21	main()

08:45 PM, 2005 Mar 22 (TC: 2_parallel_win32.exe) (ID=1) 1st Access

Stack Trace: MyThreadFunc "2_parallel_win32.cpp": 16

Address	Line	Source
0x108E	13	long factor = 3;
0x1095	14	while (number % factor) factor += 2;
0x10C4	15	if (factor == number)
0x10CE	16	primes[number of primes++] = number;
	17	}
0x10E9	18	ExitThread(0); return 0;
	19	}
	20	

Context Definition 1st Access Stack Traces

Example: Prime Number Generation

```
#include <windows.h>
#include <stdio.h>

const long N = 1000;
long primes[N], number_of_primes = 0;
CRITICAL_SECTION primes_cs;

typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
```

```
#include <windows.h>
#include <stdio.h>
```

```
const long N = 1000;
long primes[N], number_of_primes = 0;
CRITICAL_SECTION primes_cs;
```

```
typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
{
```

```
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    InitializeCriticalSection( &primes_cs );
    for ( int thread = 0; thread < num_threads; thread++ )
    {
        pThreadArgs[thread].blocksize = N / num_threads;
        pThreadArgs[thread].seed = thread * pThreadArgs[thread].blocksize;
        pThreadArgs[thread].limit = N;
        pThreadArgs[thread].stride =
            num_threads * pThreadArgs[thread].blocksize;

        pThreadHandles[thread] =
            CreateThread( 0, 0, MyThreadFunc, (LPVOID) (pThreadArgs+thread), 0, 0 );
    }
    WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );
    printf( "Found %d primes\n", number_of_primes );
}
```


Example: Prime Number Generation

```
#include <windows.h>
#include <stdio.h>

const long N = 1000;
long primes[N], number_of_primes = 0;
CRITICAL_SECTION primes_cs;

typedef struct { long seed, limit, stride, blocksize; } THREAD_ARGS;
```

```
#include <windows.h>
```

```
main()
{
    // Decompose computation across processors
    SYSTEM_INFO info; GetSystemInfo(&info);
    int num_threads = info.dwNumberOfProcessors;
    HANDLE* pThreadHandles = (HANDLE*)malloc(num_threads*sizeof(HANDLE));
    THREAD_ARGS* pThreadArgs =
        (THREAD_ARGS*)malloc( num_threads * sizeof(THREAD_ARGS) );

    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    InitializeCriticalSection( &primes_cs );
    for ( int thread = 0; thread < num_threads; thread++ )
    {
```

```
        }
        WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );
        printf( "Found %d primes\n", number_of_primes );
    }
}
```

Example: Prime Number Generation

```

DWORD WINAPI MyThreadFunc( LPVOID pThreadArgs )
{
    THREAD_ARGS* p = (THREAD_ARGS*)pThreadArgs;
    for ( long base = 3+p->seed; base <= p->limit; base += p->stride )
        for ( long number = base; number < base+p->blocksize; number+=2 )
        {
            long factor = 3;
            while ( number % factor ) factor += 2;
            if ( factor == number )
            {
                EnterCriticalSection( &primes_cs );
                primes[ number_of_primes++ ] = number;
                LeaveCriticalSection( &primes_cs );
            }
        }
}

```

```

InitializeCriticalSection( &primes_cs );

```

```

for ( int thread = 0; thread < num_threads; thread++ )
{

```

```

    }
    WaitForMultipleObjects( num_threads, pThreadHandles, TRUE, INFINITE );
    printf( "Found %d primes\n", number_of_primes );
}

```

Intel® Threading Tools



Intel® Thread Checker

Pinpoint notorious threading bugs like data races, stalls and deadlocks

Intel® Thread Profiler

Visualize your threads over time

Identify synchronization objects that cause delays

OpenMP*, Win32* threads, POSIX* threads

Intel® Threading Tools



Intel® Thread Checker

Pinpoint notorious threading bugs like data races, stalls and deadlocks

Intel® Thread Profiler

Visualize your threads over time

Identify synchronization objects that cause delays

OpenMP*, Win32* threads, POSIX* threads

Intel® Threading Tools



Intel® Thread Checker

Pinpoint notorious threading bugs like data races, stalls and deadlocks

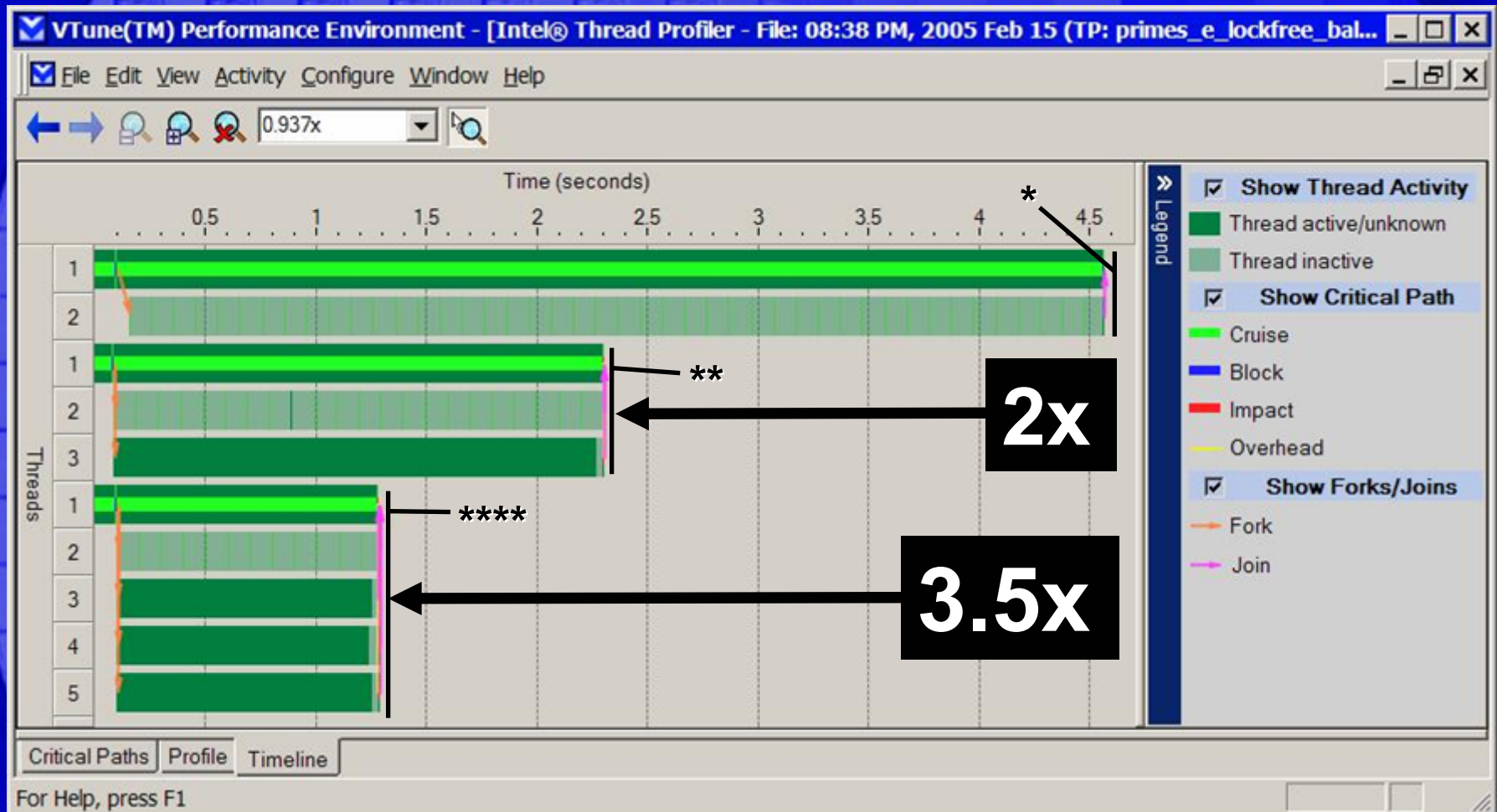
Intel® Thread Profiler

Visualize your threads over time

Identify synchronization objects that cause delays

OpenMP*, Win32* threads, POSIX* threads

Intel® Thread Profiler



* Intel® Xeon™ (HT Technology DISABLED)

** Dual-processor Intel® Xeon™ (HT Technology DISABLED)

**** Dual-processor Intel® Xeon™ (HT Technology ENABLED)

Summary

**Decompose processing across
parallel compute threads**

**OS schedules active threads to
execute on logical processors**

**Multiple execution cores ramping
across Intel platforms**

Summary

Decompose processing across parallel compute threads

OS schedules active threads to execute on logical processors

Multiple execution cores ramping across Intel platforms

Summary

Decompose processing across parallel compute threads

OS schedules active threads to execute on logical processors

Multiple execution cores ramping across Intel platforms

Summary

Decompose processing across parallel compute threads

OS schedules active threads to execute on logical processors

Multiple execution cores ramping across Intel platforms

**Create threaded apps. more quickly with OpenMP* and the Intel® Threading Tools:
www.intel.com/software/products**

Additional Resources

Thread Profiler for Win32* Threads

<https://shale.intel.com/SoftwareCollege/CourseDetails.asp?courseID=220>

Developing Multithreaded Applications: A Platform Consistent Approach

Search intel.com for “AppDevelopment.pdf”

Supported Platforms

Windows*

Microsoft Windows* XP Professional

Microsoft Windows* 2000

Microsoft Windows* Server 2003

Linux* (Remote Agent)

Red Hat* Application Server 2.1**

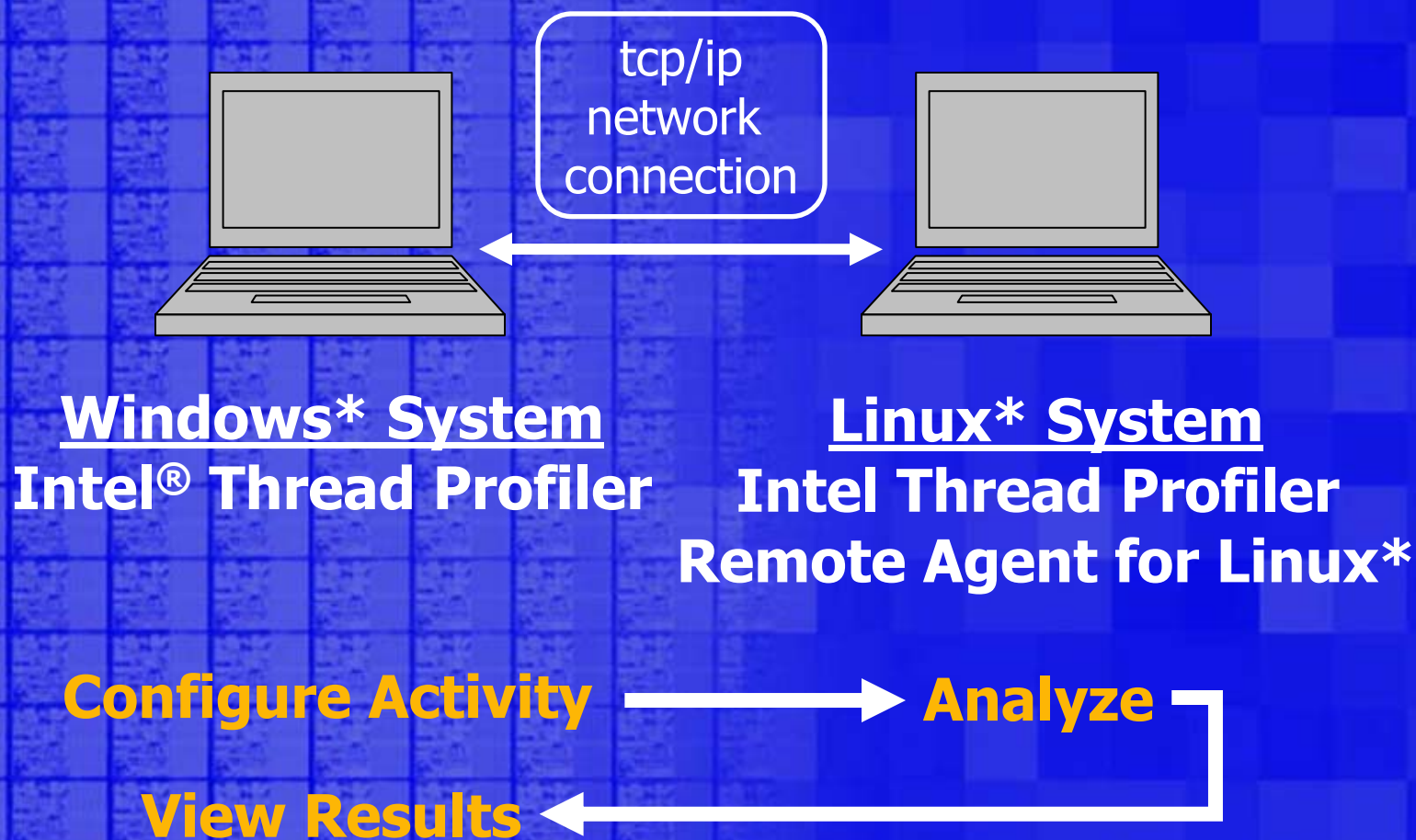
Red Hat* Enterprise Linux* 3.0

Red Hat* Enterprise Linux* 4.0

Red Flag* 4.1

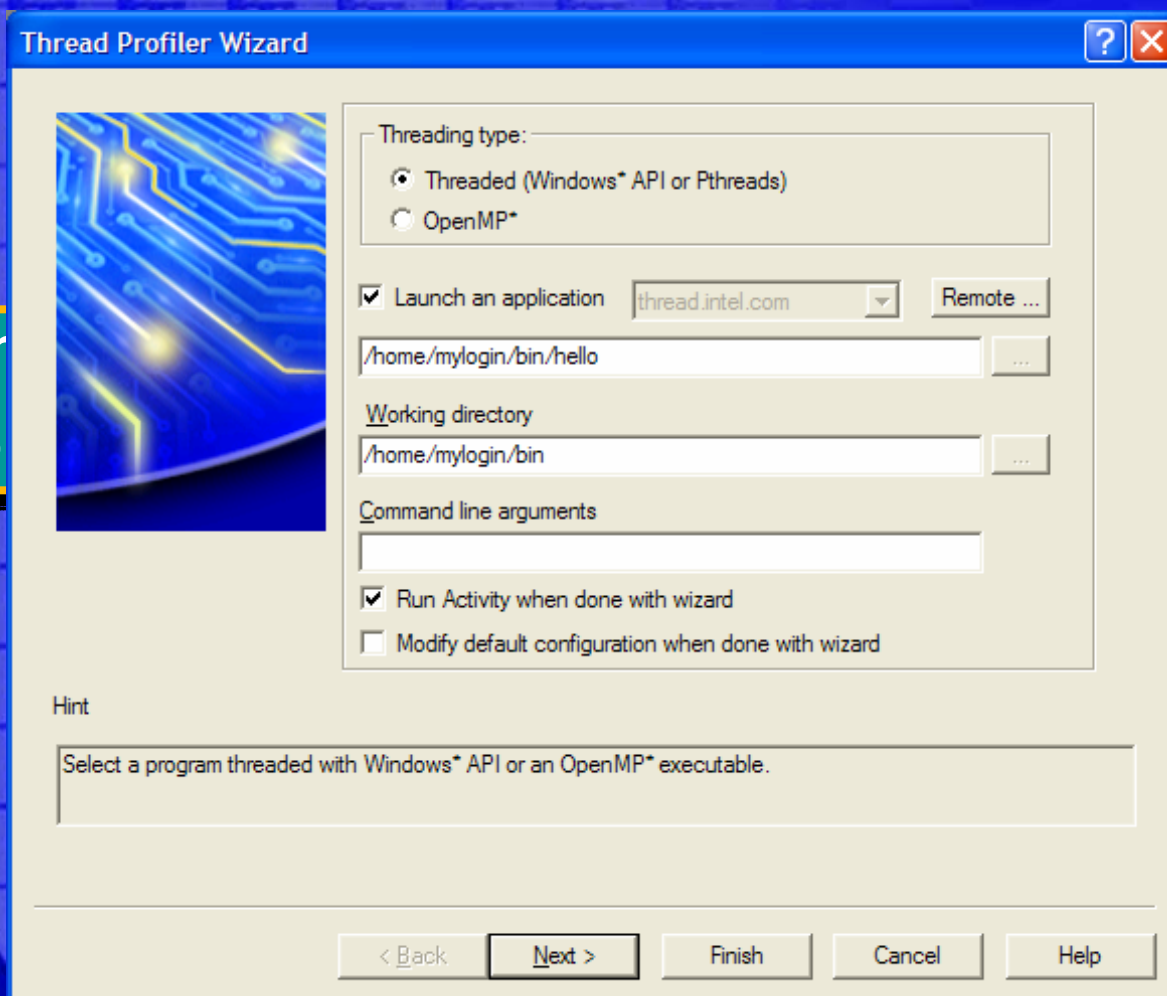
SUSE LINUX Enterprise Server* 9.0

Remote Analysis on Linux*



Remote Analysis on Linux*

Linux
ap



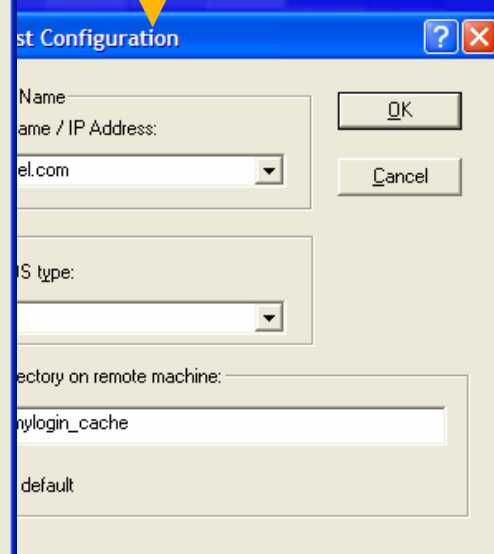
The Thread Profiler Wizard dialog box is shown with the following configuration:

- Threading type:**
 - ☒ Threaded (Windows* API or Pthreads)
 - ☐ OpenMP*
- ☒ Launch an application
 - Application: thread.intel.com
 - Remote ... button
 - Path: /home/mylogin/bin/hello
 - Working directory: /home/mylogin/bin
 - Command line arguments: (empty)
- ☒ Run Activity when done with wizard
- ☐ Modify default configuration when done with wizard

Hint:
Select a program threaded with Windows* API or an OpenMP* executable.

Navigation buttons: < Back, Next >, Finish, Cancel, Help

Linux*
system



The Host Configuration dialog box is shown with the following configuration:

- Name:** (empty)
- Name / IP Address:** thread.intel.com
- OS type:** (empty)
- Directory on remote machine:** mylogin_cache
- default:** default

Buttons: OK, Cancel

User Synchronization Support

ITT User Sync API

```
void __itt_notify_sync_prepare( void* pSyncVar );  
void __itt_notify_sync_cancel( void* pSyncVar );  
void __itt_notify_sync_acquired( void* pSyncVar );  
void __itt_notify_sync_releasing( void* pSyncVar );  
(Include libittnotify.h, Link with libittnotify.lib)
```

Usage Example

```
volatile long myLock = 0;  
__itt_notify_sync_prepare( &myLock );  
while( !InterlockedCompareExchange(&myLock, 1, 0) );  
__itt_notify_sync_acquired( &myLock );  
  
// <atomic section>  
  
__itt_notify_sync_releasing( &myLock );  
InterlockedDecrement( &myLock );
```